3. Write a program segment equivalent to the following which uses a `while`-statement instead of the `for`-statement. Your program segment must perform the same operations in the same order as the given code.

```
int i, j, num[20], sqr[20];
for (i=0, j=0; i < 20; i++, j++){
    num[i] = i;
    sqr[i] = i*j;
}
```

```
int i, j, num[20], sqr[20];

i = j = 0;
while ( i < 20)
{
    num[i] = i;
    sqr[i] = i*j;
    i++;
    j++;
}
```

C. ***(8 marks)***

Suppose we have variables declared and assigned as shown below

```
int i, j, k;
float x, y;

i = j = k = 3;
x = 0.0;
y = 2.3;
```

Complete the following table. An example is given to illustrate what needs to be provided to complete the table. The "equivalent expression" must make explicit, by use of parenthesis, the precedence of all operators. Assume that, prior to evaluation of each expression, the values of the variables are as initialized above

Expression	Equivalent Expression	Value
`i && j && k`	`(i && j) && k`	1
`y || i && j - 3`	y || (i && (j-3))	1
`i---i-j`	((i--) - i) - j	-2 or -3
`i * j && x < y`	(i*j) && (x<y)	1
`x != y && j + 1 == !k - 4`	(x != y) && ((j+1) == ((!k)+4))	1

dependent

↑

common error: not explicitly showing the precedence of all operators

A. ***(1 mark each, for 9 marks)***

For each of the statements below, indicate whether it is **true ("T")** or **false ("F")**.

F The following two statements are equivalent (the storage class and type of variable a is the same in each case): See pg 104 of K&R

```
char a[] = "C World!";
```

and

```
char *a = "C World!";
```

T A `static` variable not explicitly initialized is automatically initialized by default as if it (or its members, if it is an array) were assigned the constant 0.

T On a machine with a word size of 32 bits, variables of type `long int` and `(int *)` are of the same size

F In C, arguments which are non-pointer types are passed to a function using call-by-value Arguments which are pointer types are passed by reference. see pg 202 of K&R

T A function prototype defines the number and types of arguments which are passed to a function, and the return type of the function (type of the return value, if any)

F Comments can be nested; that is, one piece of comment (delimited by "`/*`" and "`*/`") can appear within another (also delimited by "`/*`" and "`*/`").

T The escape sequence for a carriage-return character is `\r`.

F The *lint(1)* command is used to remove extra or extraneous white space from a C program.

F To create a `.h` file (header file) from a normal `.c` file, one invokes the C compiler (`cc`) with the `-h` option. For instance,

```
cc -h myatdio.c
```

will create `myatdio.h` from `myatdio.c`

B. ***(3+2+3 = 8 marks)***

Answer each of the following questions with a concise, precise answer.

1. What is the meaning of each of the following operators? I.e. say what operation, or combination of operations, each of the following operators performs

   (a) `&` bitwise-and
   or
   address-of

   (b) `<<=`
   left shift and assign

2. Name two unconditional branch statements (constructs).

   any two of { break, continue, goto, return }

D. (3+3 = 6 marks)

Knowledge about C programming concepts is necessary to prevent errors when programming For each of the following C code segments, indicate and describe the error being made I.e. in each of the following pieces of code, a programming error is being made. Identify (in some easily discernible way) the error in each case. Also describe (by way of a short description) the nature of the error Then indicate how the error can be concisely corrected without changing the intended logic of the program sample

1.

```
#define NAME "Jane Doe",

int main( void ) (
{
    char *s;

    *s = NAME;
    printf("%s\n", NAME);
    printf("%s\n", s);
    return 0;
}
```

unintentional error made when creating the test document

note that even without #include <stdio.h> the program will compile and execute correctly

Nature of the error a value of type char (i.e. *s, since the type of s is pointer-to-char) is being assigned a value of type pointer-to-char. The fix is to rewrite the offending statement as

s = NAME;

2.

```
#include <stdio.h>
#include <string h>

main()
{
    File *infile,
    int x,

    infile = fopen( "foobar", 'r' );
    fscanf( infile, '%d', &x );
    printf( 'the number is %x', x );
}
```

Nature of the error: Identifier and symbol names in C are case specific. Thus the type identifier "FILE" is not the same as "File" Hence either the declaration of infile will be erroneous (if "File" is not a declared type) or infile will be of the wrong type

fix: change "File" to "FILE"

E. (3+1+$\overset{1}{2}$+3+3+$\overset{2}{1}$ = 13 marks)

Suppose that you are writing a C program to process a simple student information database, and produce output reports. The input file to be used by your program has up to 100 rows of data, one for each student. Each row contains six fields of data, namely the student's student number (an integer),

last name (maximum of 10 letters), first name (maximum of 10 letters), college (2 letters), year of program (an integer), and cumulative grade point average (CGPA) (a rational number), in that order. For example, the first few lines of the input file might look like this

```
123456 Smith Chris      AR  3 84 0
192765 Balloo Mogley    AR  4 74.9
612076 Chang Tzue       EN  4 81.2
742806 Chretien Jeanne  CO  3 87.1
812712 Baggins Bilbo    GS 15 64.6
904615 Wang Yanwei      EN  4 91.0
```

Assume that your program must read in and store this information for subsequent processing (e g., sorting by last name or by CGPA)

(a) Below, show the declaration of a `struct StudentRecord` that is capable of storing one row of the above information.

```
struct StudentRecord {
    int StudentNumber;
    char LastName[10];
    char FirstName[10];
    char College[2];
    int Year;
    float CGPA;
};
```

(b) In the space below, define the manifest constant MAX_STUDENTS which is the maximum number of `StudentRecord` structures that may be necessary (in your program) to hold all the data in the input file

#define MAX.STUDENTS 100

see pg 230 of KER for defn of "manifest constant"

(c) Show the declaration of an array, `studentlist`, of `StudentRecord` structures (assuming the `StudentRecord` type as declared in part (a)) The array `studentlist` is capable of storing up to MAX_STUDENTS student records

struct StudentRecord studentlist[MAX_STUDENTS];

max 1 mark

(d) Show the declaration of a variable `studentrecp` which is a pointer to a record of type `StudentRecord` (defined in part (a)). Then show the allocation of a `StudentRecord struct` from dynamic-allocated memory, and the assignment to `studentrecp` of the pointer to this memory. Make sure there are no type errors and make all type castings explicit.

struct StudentRecord *studentrecp;

studentrecp = (struct StudentRecord *) malloc(sizeof(struct StudentRecord));

2. Give at least two advantages and two disadvantages of using macros instead of functions. You can use examples to help make your point(s) clear

common errors: not comparing with functions, i.e. giving characteristics (of using macros) that are shared with (many) functions

two advantages (chosen from amongst many possible)

- macros are more efficient at runtime because they do not require the overhead of function call and return
- a macro can be used in places where function calls would be disallowed, e.g

```
#define NUM 5          vrs.     int NUM(){
                                    return(5);
char c[NUM];                    }
                                   :
                                char c [NUM];
```

two disadvantages (chosen from amongst many possible) are given in the K&R text, pg 90

(e) **Show the declaration of a variable `studentreclist` which is a pointer to a dynamically allocated array of pointers, each of which points to a `StudentRecord struct`. Then show the statement necessary to allocate storage for an array of size `MAX_STUDENTS`, where each element of the array is a pointer to a `StudentRecord struct`, and assign to `studentreclist` the pointer to the dynamically allocated storage. Make sure there are no type errors and make all type castings explicit. Finally, make the 0th row of `studentreclist` be the record pointed to by `studentrecp` (from part (d))**

```
struct StudentRecord **studentreclist;
/* cannot be  struct StudentRecord *studentreclist[]; */

studentreclist = (struct StudentRecord **) calloc (MAX_STUDENTS,
                                   sizeof (struct StudentRecord *));

studentreclist[0] = studentrecp;
```

(f) **Suppose that records for students have been read in from the file, and information stored in dynamically allocated storage pointed to by `studentreclist`. Suppose that `c` is declared to be of type `char`. Give a statement which will assign to `c` the value of the 3rd character of the last name of the fifth student. Remember that in C indexing begins at 0.**

2 marks

```
c = studentreclist[4] -> LastName[2];
```

F. ***(2+4 = 6 marks)***

Answer each of the following questions with a concise answer

1. **Give two reasons why the C language became extremely popular, and continues to be popular more than 25 years after its inception.**

common errors: not giving a complete reason, but only giving the first part of an argument

two possible answers (from amongst many)

- has high-level constructs, yet can perform almost any low-level manipulation possible with assembly language
- there are high-quality open source compilers for the language